PATENTS • DESIGNS • COPYRIGHT • TRADE MARKS
The Patent Office

INVESTOR IN PEOPLE

The Patent Office
Concept House
Cardiff Road
Newport
South Wales
NP10 8QQ

I, the undersigned, being an officer duly authorised in accordance with Section 74(1) and (4) of the Deregulation & Contracting Out Act 1994, to sign and issue certificates on behalf of the Comptroller-General, hereby certify that annexed hereto is a true copy of the documents as originally filed in connection with the patent application identified therein.

In accordance with the Patents (Companies Re-registration) Rules 1982, if a company named in this certificate and any accompanying documents has re-registered under the Companies Act 1980 with the same name as that with which it was registered immediately before re-registration save for the substitution as, or inclusion as, the last part of the name of the words "public limited company" or their equivalents in Welsh, references to the name of the company in this certificate and any accompanying documents shall be treated as references to the name with which it is so re-registered.

In accordance with the rules, the words "public limited company" may be replaced by p.l.c., plc, P.L.C. or PLC.

Re-registration under the Companies Act does not constitute a new legal entity but merely subjects the company to certain additional company law rules.

Signed: M Cook

Dated: 29 November 2001

Patents Form 1/77

Patents Act 1977
(Rule 16)

08 NOV 2000
RECEIVED BY HAND

The Patent Office

1/77

09NOV00 E582239-7 D02847
P01/7700 0.00-0027294.8

Request for grant of a patent

(See the notes on the back of this form. You can also get an explanatory leaflet from the Patent Office to help you fill in this form)

The Patent Office

Cardiff Road
Newport
Gwent NP9 1RH

1. Your reference — HL77291/MIW

2. Patent application number *(The Patent Office will fill in this part)* — 0027294.8 — 8 NOV 2000

3. Full name, address and postcode of the or of each applicant *(underline all surnames)*

SIROYAN LIMITED
12 - 15 Hanger Green Lane
London
W5 3AY
UK

Patents ADP number *(if you know it)* — 7830763003

If the applicant is a corporate body, give the country/state of its incorporation — United Kingdom

4. Title of the invention

PROCESSORS AND METHODS OF OPERATING PROCESSORS

5. Full name of your agent *(if you have one)* — Haseltine Lake & Co.

"Address for service" in the United Kingdom to which all correspondence should be sent *(including the postcode)*

Imperial House
15-19 Kingsway
London WC2B 6UD

Patents ADP number *(if you know it)* — 34001

6. If you are declaring priority from one or more earlier patent applications, give the country and the date of filing of the or of each of these earlier applications and *(if you know it)* the or each application number

Country	Priority application number *(if you know it)*	Date of filing *(day/month/year)*

7. If this application is divided or otherwise derived from an earlier UK application, give the number and the filing date of the earlier application

Number of earlier application	Date of filing *(day/month/year)*

8. Is a statement of inventorship and of right to a grant of patent required in support of this request? *(Answer "Yes" if:*
a) any applicant named in part 3 is not an inventor, or
b) there is an inventor who is not named as an applicant, or
c) any named applicant is a corporate body.
See note (d))

Yes

9. Enter the number of sheets for any of the following items you are filing with this form. Do not count copies of the same document.

Continuation sheets of this form	
Description	24
Claims(s)	5
Abstract	1
Drawing(s)	8

10. If you are also filing any of the following, state how many against each item.

Priority documents	
Translations of priority documents	
Statement of inventorship and right to a grant of patent *(Patents Form 7/77)*	3
Request for preliminary examination and search *(Patents Form 9/77)*	1
Request for substantive examination *(Patents Form 10/77)*	1
Any other documents *(please specify)*	

11. I/We request the grant of a patent on the basis of this application.

Haseltine Lake & Co. Agents for the Applicants	Signature	Date
	Haseltine Lake & Co.	8 November 2000

12. Name and daytime telephone number of person to contact in the United Kingdom — Michael Williams — [020] 7420 0500

Warning

After an application for a patent has been filed, the Comptroller of the Patent Office will consider whether publication or communication of the invention should be prohibited or restricted under Section 22 of the Patents Act 1977. You will be informed if it is necessary to prohibit or restrict your invention in this way. Furthermore, if you live in the United Kingdom, Section 23 of the Patents Act 1977 stops you from applying for a patent abroad without first getting written permission from the Patent Office unless an application has been filed at least 6 weeks beforehand in the United Kingdom for a patent for the same invention and either no direction prohibiting publication or communication has been given, or any such direction has been revoked.

Notes

a) *If you need help to fill in this form or you have any questions, please contact the Patent Office on 0645 500505.*

b) *Write your answers in capital letters using black ink or you may type them.*

c) *If there is not enough space for all the relevant details on any part of this form, please continue on a separate sheet of paper and write "see continuation sheet" in the relevant part(s). Any continuation sheet should be attached to this form.*

d) *If you have answered `Yes' Patents Form 7/77 will need to be filed.*

e) *Once you have filled in the form you must remember to sign and date it.*

f) *For details of the fee and ways to pay please contact the Patent Office.*

Patents Form 7/77

Patents Act 1977
(Rule 15)

THE PATENT OFFICE
08 NOV 2000
RECEIVED BY HAND

The Patent Office

7/77

Statement of inventorship and of right to grant of a patent

The Patent Office

Cardiff Road
Newport
Gwent NP9 1RH

1. Your reference HL77291/MIW

2. 0027294.8 8 NOV 2000

3. Full name of the or of each applicant SIROYAN LIMITED

4. Title of the invention

PROCESSORS AND METHODS OF OPERATING PROCESSORS

5. State how the applicant(s) derived the right from the inventor(s) to be granted a patent

By virtue of the inventors' contracts of employment

6. How many, if any, additional Patent Forms 7/77 are attached to this form?
(see note (c)) None

7. I/We believe that the person(s) named over the page *(and on any extra copies of this form)* is/are the inventor(s) of the invention which the above patent application relates to.

Signature Date

Haseltine Lake & Co. Agents for the Applicants Haseltine Lake & Co 8 November 2000

8. Name and daytime telephone number of person to contact in the United Kingdom Michael Williams [020] 7420 0500

Notes

a) *If you need help to fill in this form or you have any questions, please contact the patent office on 0645 500505.*

b) *Write your answers in capital letters using black ink or you may type them.*

c) *If there are more than three inventors, please write the names and addresses of the other inventors on the back of another Patents Form 7/77 and attach it to this form.*

d) *When an application does not declare any priority, or declares priority from an earlier UK application, you must provide enough copies of this form so that the Patent Office can send one to each inventor who is not an applicant.*

e) *Once you have filled in the form you must remember to sign and date it.*

Enter the full names, addresses and postcodes of the inventors in the boxes and underline the surnames

Kar-Lik Kasim WONG
4 The Avenue
Old Windsor
Berkshire
SL4 2RS
UK

8017964001

Patents ADP number *(if you know it)*

Nigel Peter TOPHAM
6 Carolina Place
Finchampstead
Wokingham RG40 4PQ
UK

7830797002

Patents ADP number *(if you know it)*

Patents ADP number *(if you know it)*

Reminder
Have you signed this form?

PROCESSORS AND METHODS OF OPERATING PROCESSORS

The present invention relates to processors and methods of operating processors. The invention has particular application in parallel pipelined processors, such as very long instruction word (VLIW) processors.

High performance processors use a technique known as pipelining to increase the rate at which instructions can be processed. Pipelining works by executing an instruction in several phases, with each phase being executed in a single pipeline stage. Instructions flow through successive pipeline stages, with all partially-completed instructions moving one stage forward on each processor clock cycle. Instructions complete execution when they reach the end of the pipeline.

Processors attempt to keep pipelines full at all times, thus ensuring a high rate of instruction completion. However, it is possible that an instruction may not be able to progress through one of the stages of a pipeline in a single clock cycle for some reason, for example, because it needs to access slow memory or to compute a multi-cycle operation. Such an event is known as a stall. When stage i of a pipeline stalls it prevents the instruction at stage i-1 from making forward progress, even if the instruction at stage i-1 is not itself stalled. This in turn stalls stage i-2, and so on up to stage 0 (the first stage). When there is a stall at stage i, a signal flows to all stages from 0 to i-1 in the pipeline to cause them to stall before the next active edge of the pipeline clock.

Some processor architectures provide two or more parallel pipelines for processing different instructions (or different parts of an instruction)

simultaneously. In this case, the stall signal must be distributed to all pipelines to ensure that instructions which are issued in parallel also complete in parallel. However, the delay of propagating such a global stall signal may restrict the operating clock frequency of the processor. Furthermore, the distance such a signal would have to travel would grow as more pipelines were added. Hence a processor having more pipelines would need a lower clock frequency, thus defeating the high throughput objective of adding further pipelines.

The present invention seeks to overcome the above disadvantages.

According to a first aspect of the present invention there is provided a processor comprising:

a plurality of pipelines, each pipeline having a plurality of pipeline stages for executing an instruction on successive clock cycles; and

stalling means for stalling the execution of instructions in all of the pipelines in response to a stall signal generated in any one of the pipelines;

wherein the stalling means is adapted to stall the execution of instructions in different pipelines in different clock cycles in response to the stall signal.

Stalling the execution of instructions in different pipelines in different clock cycles may make additional time available for distributing signals, such as global stall signals. This may allow the processor to operate at a higher speed than would otherwise be the case.

Preferably the stalling means is adapted to stall the execution of an instruction in a pipeline not generating the stall signal at least one clock cycle

later than the execution of an instruction in a pipeline generating a stall signal. This may allow at least one clock cycle for distributing the stall signal from the pipeline generating the stall signal to other pipelines.

Preferably the stalling means is adapted to release the stall in the pipeline not generating the stall signal at least one clock cycle later than the stall in the pipeline generating the stall signal. In this way, the instructions in the various pipelines may return to alignment after being out of step for one or more clock cycles.

If two or more pipelines generate a stall signal simultaneously, then it may not be necessary to stall each pipeline in response to the stall signal generated by the other pipeline. This is because each pipeline may have already implemented the appropriate stall. If each pipeline were also stalled in response to the stall signal generated by the other pipeline, more stalls than necessary may be implemented. Thus the stalling means may be arranged such that, when a pipeline stage in a first pipeline receives a stall signal from a second pipeline, the execution of instructions in the pipeline stage is not stalled if the pipeline stage stalled in the previous cycle in response to a stall signal generated by the first pipeline.

The stalling means may be arranged such that, when a pipeline generates a stall signal at a stage i, all stages up to and including stage i of that pipeline are stalled. In this way, earlier stages in a pipeline are stalled to prevent instructions in a pipeline from overwriting each other. Later stages need not be

stalled, so the instructions in later stages may continue to progress through the pipeline as normal, although, if required, some or all of the later stages could also be stalled.

The stalling means may be arranged such that, when a pipeline generates a stall signal at a stage i, all stages up to and including stage i of that pipeline are stalled on a given clock cycle, and all stages up to and including stage i+m of a pipeline not generating a stall signal are stalled m clock cycles later than said given clock cycle, where m is an integer greater than or equal to 1. By providing that all stages up to i+m are stalled m clock cycles later in a pipeline not generating a stall signal, the instructions in that pipeline which correspond to the stalled instructions in the pipeline generating the stall signal are stalled.

The processor may comprise a plurality of pipeline clusters, each cluster comprising a plurality of pipelines. In this case, the stalling means may be arranged to stall execution of instructions in pipelines within a cluster in the same clock cycle.

Preferably, in operation, instructions entering the respective pipelines in parallel (that is to say, in a particular clock cycle) exit the pipelines in parallel.

The present invention also has application in the distribution of signals other than stall signals, and thus, according to another aspect of the invention, there is provided a processor comprising a plurality of pipelines, each pipeline having a plurality of pipeline stages for executing an instruction on successive clock cycles, the processor being adapted to allow two

instructions which enter and exit respective pipelines in parallel to become out of step with each other for at least one clock cycle.

According to another aspect of the invention there is provided a processor comprising a plurality of pipelines, each pipeline having a plurality of pipeline stages, at least some pipeline stages having a processing circuit arranged to perform the following functions:

receiving a processor clock signal;

processing instructions on successive cycles of said processor clock signal;

generating a stall signal when the processing circuit requires the processing of an instruction to be stalled; and

stalling the processing of an instruction in response to an externally-generated stall signal;

the processor also including control logic arranged to cause the execution of instructions in different pipelines to stall in different cycles in response to the same stall signal.

Analogous method aspects of the invention are also provided. Apparatus features of the invention may be applied to method aspects and *vice versa*.

Preferred features of the present invention will now be described, purely by way of example, with reference to the accompanying drawings, in which:-

Figure 1 is a block diagram of parts of a processor according to a first embodiment of the present invention;

Figure 2 is a representation of a seven stage pipeline;

Figure 3 shows parts of two stages of first and second pipelines in the Figure 1 processor;

Figure 4 is a block diagram of parts of a processor according to a second embodiment of the present invention;

Figure 5 shows parts of one stage of a pipeline in the Figure 4 processor;

Figure 6 is a state transition diagram showing the operation of part of the pipeline stage of Figure 5;

Figures 7 to 10 show one example of the operation of the second embodiment; and

Figures 11 to 14 show another example of the operation of the second embodiment.

Figure 1 is a block diagram of parts of a processor according to a first embodiment of the present invention. In this embodiment, the processor is a very long instruction word (VLIW) processor which is designed to execute long instructions which may divided into smaller instructions.

Referring to Figure 1, processor 1 comprises instruction issuing unit 10, schedule storage unit 12, first and second execution units 14, 16, and first and second register files 18, 20. The instruction issuing unit 10 has two issue slots IS1, IS2 connected respectively to the execution units 14, 16. The first execution unit 14 is connected to the first register file 18 and the second execution unit is connected to the second register file 20. The register files 18, 20 are connected to each other via a bus 22. As an alternative to two register files connected via a bus, a single register file could be provided instead. Each of the execution units 18, 20 is also connected to an external memory 24 via bus 26. In this example the external memory 24 is random access memory (RAM),

although it may be any other type of memory.

In operation, an instruction packet for execution is passed from the schedule storage unit 12 to the instruction issuing unit 10. The instruction issuing unit 10 divides the instruction packet into its constituent instructions, and issues the two instructions to the execution units 14, 16 via the issue slots IS1 and IS2 respectively. The execution units 14, 16 then execute the various instructions simultaneously. In this way, different parts of a long instruction are processed in parallel.

Each of the execution units 14, 16 uses a pipelining technique to maximise the rate at which it processes instructions. Pipelining works by implementing each of a plurality of phases of instruction execution as a single pipeline stage. Instructions flow through successive pipeline stages, in a production-line fashion, with all partially-completed instructions moving one stage forward on each processor clock cycle. Instructions complete execution when they reach the end of the pipeline.

Figure 2 is a representation of a seven stage pipeline. In this representation, the content of each stage is the sequence number of the instruction which has reached that stage of the pipeline. Instructions in the pipeline flow from left to right, from stages 0 to 7.

It is desirable to keep the pipeline full at all times, thus ensuring a high rate of instruction completion. However, it is possible that an instruction may not be able to progress through one of the stages in a single clock cycle for some reason, for example because it

needs to access slow memory or to compute a multi-cycle operation. Such an event is known as a stall. When stage i stalls it prevents the instruction at stage i-1 from making forward progress, even if the instruction at stage i-1 is not itself stalled. This in turn stalls stage i-2 and so on up to stage 0 (the first stage). If stages 0 to i are stalled at time T, then at time T+1 stage i+1 will have no instruction to process. If the stall persists for another cycle, then at time T+2 stages i+1 and i+2 and will have no instructions to process. These empty pipeline stages are known as bubbles. In Figure 2, a two-cycle bubble is shown in stages 3 and 4.

VLIW processors are designed such that instructions which are issued to different pipelines in parallel also complete their execution in parallel. If this rule were relaxed it would prove very difficult to stop a running process cleanly and to restart it at some later time. Thus, if there is a stall in one pipeline, each other pipeline must also stall to ensure that the various instructions exit the pipelines in parallel. For example, if there is a stall in the pipeline in execution unit 14 in Figure 1, then the pipeline in execution unit 16 must also stall.

When there is a stall at stage i of a pipeline, a stall signal is generated by that stage. This stall signal is distributed to all stages from 0 to i-1 in the pipeline to cause them to stall before the next active edge of the pipeline clock.

One possible scheme for stalling the other pipeline in the Figure 1 processor would be to take the logical OR of the stall signal from each stage of both pipelines and to distribute the result to both pipelines.

However, this would require the transmission of a global stall signal to all pipeline stages before the next active edge of the pipeline clock. For high speed processors the delay in propagating such a global stall signal may restrict the operating clock frequency of the processor. Furthermore, if more pipelines were added in order to increase the processing rate, the physical distance the global stall signal would have to travel would increase, thereby restricting the operating clock frequency even further.

According to the first embodiment, instructions passing through the respective pipelines may get one stage out of step with each other, thus providing a full clock period for a stall signal to pass from one pipeline to the other.

The operation of the first embodiment will now be described with reference to Figure 3. Figure 3 shows two stages, stage i and stage i+1, of a first pipeline (pipeline 1) and the corresponding stages of a second pipeline (pipeline 2). Stage i of pipeline 1 comprises instruction register 40, processing circuit 42, OR gate 44, registers 45, 46, AND gate 47 and OR gate 48; stage i+1 of pipeline 1 comprises instruction register 50, processing circuit 52, OR gate 54, registers 55, 56, AND gate 57 and OR gate 58; stage i of pipeline 2 comprises instruction register 60, processing circuit 62, OR gate 64, registers 65, 66, AND gate 67 and OR gate 68; stage i+1 of pipeline 2 comprises instruction register 70, processing circuit 72, OR gate 74, registers 75, 76, AND gate 77 and OR gate 78. In this example, registers 45, 55, 65, 75 and 46, 56, 66, 76 are implemented as D-type flip-flops. All clock inputs are fed by a common clock signal. Stages i and i+1 of pipeline 1 are part of the execution unit 14 in Figure

1 and stages i and i+1 of pipeline 2 are part of the execution unit 16 in Figure 1.

In normal operation, each processing circuit 42, 52, 62, 72 executes one phase of an instruction held in the corresponding register 40, 50, 60, 70. The stage i processing circuits 42 and 62 execute in parallel phase i of two instructions belonging to one VLIW packet and the stage i+1 processing circuits 52 and 72 execute simultaneously phase i+1 of two instructions belonging to another VLIW packet. On each clock cycle, the instructions held in the registers are passed to the next registers in the pipelines for further processing. In this way instructions flow through the pipelines with all partially completed instructions moving forward one stage on each clock cycle.

Each of the processing circuits 42, 52, 62, 72 is able to assert a stall signal if it requires the progress of the instruction it is executing to be stalled on the next clock cycle. The stall signals from processing circuits 42, 52, 62, 72 are fed to OR gates 44, 54, 64, 74 respectively. The outputs of the OR gates 44, 54, 64, 74 are fed to OR gates 48, 58, 68, 78 respectively. The OR gates 48, 58, 68, 78 output hold signals to processing circuits 42, 52, 62, 72 respectively. Thus, if one of the processing circuits asserts a stall signal, the hold signal of that processing circuit is asserted via the corresponding OR gates. If the hold signal input to a processing circuit is set, then that processing circuit will stall on the next clock cycle.

The output of each of the OR gates 44, 54, 64, 74 is also fed as a ripple signal to the corresponding OR gate in the previous stage in the same pipeline. The ripple signals thus ripple down the pipelines, so that

if a processing circuit asserts a stall signal, the hold signals input to all previous processing circuits in the same pipeline are also asserted.

The output of each of the OR gates 44, 54, 64, 74 is also fed to the next stage of the other pipeline as a global signal. Each of the registers 46, 56, 66, 76 receives such a global signal from the previous stage of the other pipeline. For example, the output of OR gate 44 is fed to register 76 and the output of OR gate 64 is fed to register 56.

Each of the registers 46, 56, 66, 76 delays the signal at its input until the next clock cycle. Thus, the output of each of the registers 46, 56, 66, 76 is the global signal from the previous stage of the other pipeline, delayed until the next clock cycle. The outputs of the registers 46, 56, 66, 76 are fed via respective AND gates 47, 57, 67, 77 to respective OR gates 48, 58, 68, 78, which output hold signals to processing circuits 42, 52, 62, 72 respectively. Thus, assuming the other inputs to the AND gates 47, 57, 67, 77 are set, if a processing circuit asserts a stall signal, the hold signal input to the next processing circuit in the other pipeline will be asserted in the next clock cycle.

The stall signals of processing circuits 42, 52, 62, 72 are also fed to registers 45, 55, 65, 75 respectively. The inverting outputs of registers 45, 55, 65, 75 are fed to AND gates 47, 57, 67, 77 respectively. Thus, the output of each of the AND gates 47, 57, 67, 77 is reset if the corresponding processing circuit asserted a stall signal in the previous clock cycle, regardless of the state of registers 46, 56, 66, 76. This prevents the registers 46, 56, 66, 76 from generating a

stall signal if the corresponding pipeline stage stalled in the previous clock cycle due to a locally generated stall signal. In this way, if the two pipelines generate a stall signal in the same clock cycle, only one stall is implemented.

As an example, if processing circuit 42 sets the stall signal in a given clock cycle, then OR gates 44 and 48 ensure that the hold signal of that stage is set. A ripple signal flows to all stages i-1 to 0 via the corresponding OR gates of those stages, causing each of those stages to set the hold signal. Thus, a hold signal is set at all stages 0 to i of the first pipeline before the next active edge of the pipeline clock, causing those stages to stall on the next clock cycle. However, because the output of OR gate 44 is delayed by register 76, the hold signal input to processing circuit 72 is not set until the next but one active edge of the pipeline clock. Similarly, the hold signal input to processing circuit 62 is delayed by one clock cycle due to register 66, and so on until stage 1 of the second pipeline. Thus, stages 0 to i+1 of the second pipeline stall one clock cycle later than stages 0 to i of the first pipeline. It is necessary to stall stages 0 to i+1 of the second pipeline because the instructions in that pipeline will have advanced one stage while the instructions in the first pipeline are stalled. Stage 0 of a pipeline is arranged to stall when stage 1 of that pipeline stalls.

While the stall signal from processing circuit 42 remains set, the hold signal to stages 0 to i of the first pipeline and stages 0 to i+1 of the second pipeline remain set, causing those stages to remain stalled. When the stall signal is released, the hold signals to stages 0 to i of the first pipeline are also

released before the next active edge of the clock, so that normal operation resumes in that pipeline on the next clock cycle. However, due to the operation of the registers 66, 76 (and the corresponding registers in previous pipeline stages), the hold signals to stages 0 to i+1 of the second pipeline are only released on the next but one clock cycle. Thus in the second pipeline normal operation resumes on the next but one clock cycle. This one cycle delay in releasing the stall allows the two pipelines to get back into step with each other.

Thus it will be seen that individual instructions within the two pipelines may become out of step by one cycle, thereby providing a full clock period for stall signals to propagate between the pipelines.

The first embodiment has the following main features:

1. Either of the pipelines can assert a stall signal in any cycle that is not itself the subject of a stall.
2. Stages prior to a stalling stage that are in the same pipeline will stall in the same cycle as the stalling stage, but stages in the other pipeline will be stalled one cycle later.
3. When a stall is released, all stalled stages in that pipeline make forward progress immediately, but stalled stages in the other pipeline are released one cycle later. The net effect of this is that the stalled operations return to alignment after being out-of-step by one stage during the stall.
4. The logic to control the stalling of each stage is local to that stage. A full clock cycle is available to distribute the global stall signal to the other pipeline.

It will be noted that the last stage of a pipeline is not able to assert a stall signal, because it would not then be able to stall the other pipeline in time to prevent the instruction in the last stage from exiting the pipeline. This may be dealt with, either by arranging the processor and the instruction set such that the last stage of a pipeline never needs to assert a stall signal, or by adding a final dummy stage to each pipeline.

If desired, a delay of two or more clock cycles may be introduced for the propagation of stall signals between the pipelines, by arranging the registers 45, 55, 65, 75 and 46, 56, 66, 76 to delay the signals at their inputs by two or more clock cycles. For example, two or more D-type flip-flops connected in series may be used in place of each flip-flop 45, 46, 55, 56, 65, 66, 75, 76 in Figure 3. In this case, the processor may ensure that the last m stages of the pipeline do not assert a stall signal, where m is the number of clock cycles delay, or a number m of dummy stages may be added, or a combination of the two approaches may be used.

Figure 4 shows parts of a processor according to a second embodiment. Referring to Figure 4, processor 100 comprises an instruction issuing unit 102, a schedule storage unit 104, first to eighth execution units (E.U.) 106, 108, 110, 112, 114, 116, 118, 120 and first to fourth register files 122, 124, 126, 128. The instruction issuing unit 102 has eight issue slots IS1 to IS8 connected respectively to the execution units 106 to 120. The first and second execution units 106, 108 are connected to the first register file 122, the third and fourth execution units 110, 112 are connected to the second register file 124, the fifth and sixth

execution units 114, 116 are connected to third register file 126, and the seventh and eighth execution units 118, 120 are connected to the fourth register file 128. Each of the execution units is also connected to an external memory 132, such as a RAM device, via bus 134.

In operation, an instruction packet (VLIW packet) for execution is passed from the schedule storage unit 104 to the instruction issuing unit 102. The instruction issuing unit 102 divides the instruction packet into its constituent instructions, and issues the instructions to the execution units 106 to 120 via issue slots IS1 to IS8. The execution units 106 to 120 then execute the various instructions belonging to the packet simultaneously.

The execution units 106 to 120 are divided into four groups, with each group having its own register file. This is done in order to reduce the number of access slots to any one register file. If a single register file were provided, the register file may have too many access slots, which would increase access time to the register file. Each group of execution units with a common register file may be referred to as a cluster. In Figure 4, a first cluster is formed by execution units 106, 108 and register file 122, a second cluster is formed by execution units 110, 112 and register file 124, a third cluster is formed by execution units 114, 116 and register file 126, and a fourth cluster is formed by execution units 118, 120 and register file 128. If a value held in one cluster is required in another cluster, the value is transferred between the clusters via a bus 130. While four clusters are shown in Figure 4, more or fewer clusters may be provided as required. Each cluster may have one, two or more

execution units.

As in the first embodiment, each of the execution units 106 to 120 uses pipelining to increase the rate at which it processes instructions. Pipelines within a cluster stay in step with each other, whereas pipelines in different clusters are allowed to get one or more cycles out of step with each other.

Figure 5 shows the block structure of the ith stage in a two pipeline cluster. Stage i of the cluster comprises stage i of the first pipeline in the cluster (pipeline 0), stage i of the second pipeline in the cluster (pipeline 1), and common control circuitry for controlling the two pipeline stages. Stage i of pipeline 0 comprises register 140 and processing circuit 142, and stage i of pipeline 1 comprises register 144 and processing circuit 146. The control circuitry is formed by stall control logic 148, registers 150, 152, 154 and OR gate 156.

In operation, stage i of the cluster can be either active, or stalled by a stage j in the same cluster, where $j \geq i$, or stalled by a stage l in another cluster where $l \geq i-1$. Hence, stage i of the cluster has a state variable CurrentState i, which indicates whether the stage is active (A), locally stalled (L) or globally stalled (G).

The value of the variable CurrentState i is held in register 150. The input to register 150 is a signal NextState i which determines which state the stage will be in in the next clock cycle.

The behaviour of the stall control logic 148 is governed by a set of state transitions, as shown in

Table 1. In Table 1, each row represents a possible transition that can be made and explicitly states the conditions under which it can take place. Each transition is from the current state to the next state, and will take place if the stage in question is in the current state for a given transition and the values of the boolean variables local and global_in are as indicated in the entry for that transition. The local variable is set when there is a local stall, i.e. one that originates in the local cluster. This is true whenever the current stage has a pending stall or if any stage after the current stage will be in the locally stalled state in the next cycle. The global_in variable is set if any other cluster asserted its local signal in the previous cycle.

Table 1

Transition	Current State	global_in	local	Next State
1	A	0	0	A
4	A	0	1	L
6	A	1	0	G
6	A	1	1	G
5	L	0	0	A
3	L	0	1	L
5	L	1	0	A
3	L	1	1	L
7	G	0	0	A
8	G	0	1	L
2	G	1	0	G
2	G	1	1	G

A state transition diagram for the behaviour of each pipeline stage is shown in Figure 6. In Figure 6,

transition numbers correspond to the numbers in the first column of Table 1. As will be apparent to the skilled person, the appropriate logic circuits may be derived routinely from the state transition diagram and/or table, and are accordingly not described specifically herein.

In operation, register 140 holds instructions for execution by processing circuit 142 and register 144 holds instructions for execution by processing circuit 146. If the phase of an instruction being executed in clock cycle T by one of the processing circuits 142, 146 in Figure 5 requires more than one cycle for execution, then that processing circuit asserts a stall signal. The stall signals from stage i of both pipelines are fed to OR gate 156. The output of OR gate 156 is a signal stall i, which indicates whether stage i of one or more of the pipelines in that cluster has asserted a stall signal. The signal stall i is fed to stall control logic 148.

Based on the stall i signal, the stall control logic 148 generates a signal local i, as follows:

$$\text{local } i = (\text{stall } i \text{ OR ripple } i+1)$$

The signal ripple i is set whenever the next state of stage i of the cluster will be L (i.e. locally stalled). Hence:

$$\text{ripple } i = (\text{NextState } i = L)$$

The signal ripple i+1 is the corresponding signal generated by stage i+1 of the cluster.

If ripple i is asserted during cycle T, then ripple i-

1, i-2, ..., i-k will also be asserted in the same cycle as a consequence of the local stall in stage i. The ripple-down of the localised stall signal will terminate at the left-most end of the pipeline, or earlier if some stage will not be in the L stage in cycle T+1 (i.e. NextState i-k ≠ L). This could be caused by a localised stall in another cluster at stage i-k-1 which was present in cycle T-1 and caused the global_in signal at stage i-k to be asserted in cycle T.

The NextState i signal is generated based on the local i signal, the CurrentState i signal and the global_in i signal, as shown in the final column of Table 1. On the next active edge of the clock, the signal NextState i is registered in register 150, so that the signal CurrentState i in cycle T+1 is equal to the signal NextState i in cycle T.

If the value of NextState i in any clock cycle is either L or G (corresponding to locally or globally stalled) then the hold i signal is asserted. In response, the instructions in registers 142 and 144 remain the same in the next clock cycle.

The various clusters communicate their stall statuses to each via the ripple and global_in signals. The next_global_in i signal for cluster C is the logical OR of the ripple i-1 signals from all clusters except C. The next_global_in i signal is registered at the end of every clock period in register 154 to give the signal global_in i. The global_in signals for stage 0 of all clusters are always false. In each pipeline, stage 0 is arranged to stall when stage 1 of that pipeline stalls.

For example, if the phase of the instruction being executed in cycle T by a processing circuit in stage h ($h \geq i-1$) of another cluster requires more than one clock cycle, then its local h signal is asserted in cycle T. This causes the next_global_in signals of all stages 1 to h+1 in all other clusters to be asserted before the end of cycle T. This in turn causes the global_in i signal of stall control logic 148 to be asserted in cycle T+1.

The valid i signal is used to indicate that a bubble is present in stage i of the cluster. For example, if stage i-1 is stalled, then the signal valid_out i-1 is set false, indicating that there is a bubble in stage i-1. On the next clock cycle, the signal valid_out i-1 is registered by register 152 to give the signal valid i. If valid i is set false, processing circuits 142, 146 ignore the instructions in registers 140, 144.

Thus it can be seen that the second embodiment provides a distributed stalling scheme, in which each stage determines locally whether it needs to stall the processor at each cycle of program execution and asserts stall i if a stall at stage i is required. If the stall logic for stage i determines that any one of the three causes of a stall requires that stage i be stalled in the current cycle then it asserts the hold i signal. In response, stage i retains the instruction in its input register.

The second embodiment has the following main features:

1. Any pipeline stage in any cluster can assert a stall signal in any cycle that is not itself the subject of a stall.
2. Stages prior to a stalling stage that are in the same cluster will stall in the same cycle as the

stalling stage, but stages in other pipelines will be stalled one cycle later.

3. When a stall is released, all stalled stages in that cluster make forward progress immediately, but stalled stages in other clusters are released one cycle later.

4. The logic to control the stalling of each stage is local to that stage. Where stall signals are communicated globally (from one cluster to another) a full clock cycle is available to compute the global stall condition and to distribute it to all clusters.

An example of the operation of the second embodiment will now be described with reference to Figures 7 to 10. Figure 7 shows the situation where a stall occurs in stage 3 on instruction packet 103 in cluster 1 at time T=1. At this time cluster 1 stalls immediately from stage 3 to stage 0 whereas other clusters are not stalled. Figure 8 shows the situation one cycle later at T=2. Because of the one-cycle delay to inform other clusters that they should stall at stage 3, it is not until this time that the other clusters are able to be stalled. By that time instruction packet 103 has moved on to stage 4, so the stall must take effect from stage 4.

Figure 9 illustrates the situation one cycle later at time T=3 when the stall is released and cluster 1 is free to make forward progress. In cluster 1, instruction packets 103 to 106 move forward and packet 107 is inserted. Now the instruction packets in all clusters are realigned due to the delay in releasing clusters 0, 2 and 3 from their stalled state. As shown in Figure 10, one cycle later at time T=4, all clusters are released from the stall and progress continues as

normal.

Another example of the operation of the second embodiment, in which a stall signal is generated by two different clusters, will now be described with reference to Figures 11 to 14. In this example, a local stall signal is generated by both stage 3 in cluster 1 and stage 1 in cluster 2 at time T=1.

Referring to Figure 11, at time T=1, stages 3 to 0 of cluster 1 and stages 1 and 0 of cluster 2 have the local signal set and the global_in signal reset, so that, referring to Table 1 (transition 4), the next state of these stages will be locally stalled (L).

One cycle later at time T=2 (Figure 12), stages 3 to 0 of cluster 1 and stages 1 and 0 of cluster 2 are in the locally stalled state, while the other stages are in the active state (A). Bubbles thus form in stage 4 of cluster 1 and stage 2 of cluster 2. Also at time T=2, stages 4 to 0 of clusters 0, 2 and 3, and stages 2 to 0 of cluster 1, have their global_in signals set, due to the stalls that occurred in clusters 1 and 2 in the previous cycle. In this example the stalls only last for one cycle. Thus, referring to Table 1, the next state of stages 4 to 0 of clusters 0 and 3 and stages 4 and 3 of cluster 2 will be globally stalled (G) while the next state of stages 3 to 0 in cluster 1 and 1 to 0 in cluster 2 will be active (A).

At time T=3 (Figure 13) stages 4 to 0 of clusters 0 and 3 and stages 4 and 3 of cluster 2 are globally stalled. Stages 1 to 0 of cluster 2 are active, and hence the instructions that were in these stages move forward one stage to fill in the bubble that occurred in stage 2 of cluster 2 at time T=2. Since stages 3 and 4 of cluster

2 are stalled, a new bubble forms at stage 5 of cluster 2. Cluster 1 is not stalled.

At time T=4 (Figure 14) all clusters are released from the stall and progress continues as normal.

It will be noted from the above that, if a stall signal is generated in two different clusters in the same clock cycle, this only results in one bubble occurring in the pipelines. This is because, if a stage has its current state as locally stalled (L), and the local signal is not set, the next state of that stage is active (A), regardless of whether or not the global_in signal is set. This is indicated by transition 5 in Table 1 and Figure 6.

In the above examples, it is assumed that a stall is released one clock cycle after it is initiated, so that bubbles of one stage form. However, a stall may last for an indefinite period of time, and thus bubbles of two or more stages may form, depending on the number of clock cycles for which a stall lasts.

As in the first embodiment, the delayed stalling scheme of the second embodiment will not permit stalls to originate in the final stage. This may be dealt with either by arranging the processor and the instruction set such that the last stage of a pipeline never needs to assert a stall signal, or by adding a final dummy stage to each pipeline, or both.

As in the first embodiment, the delay in distributing a global stall signal between clusters may be one or more clock cycles.

Although the above description relates, by way of

example, to a VLIW processor it will be appreciated that the present invention is applicable to processors other than VLIW processors. A processor embodying the present invention may be included as a processor "core" in a highly-integrated "system-on-a-chip" (SOC) for use in multimedia applications, network routers, video mobile phones, intelligent automobiles, digital television, voice recognition, 3D games, etc.

It will be understood that the present invention has been described above purely by way of example, and modifications of detail can be made within the scope of the invention.

Each feature disclosed in the description, and (where appropriate) the claims and drawings may be provided independently or in any appropriate combination.

CLAIMS

1. A processor comprising:

a plurality of pipelines, each pipeline having a plurality of pipeline stages for executing an instruction on successive clock cycles; and

stalling means for stalling the execution of instructions in all of the pipelines in response to a stall signal generated in any one of the pipelines;

wherein the stalling means is adapted to stall the execution of instructions in different pipelines in different clock cycles in response to the stall signal.

2. A processor according to claim 1 wherein the stalling means is adapted to stall the execution of an instruction in a pipeline not generating the stall signal at least one clock cycle later than the execution of an instruction in a pipeline generating the stall signal.

3. A processor according to claim 2 wherein the stalling means is adapted to release the stall in the pipeline not generating the stall signal at least one clock cycle later than the stall in the pipeline generating the stall signal.

4. A processor according to claim 2 or 3 wherein the stalling means is arranged such that, when a pipeline stage in a first pipeline receives a stall signal from a second pipeline, the execution of instructions in the pipeline stage is not stalled if the pipeline stage stalled in the previous cycle in response to a stall signal generated by the first pipeline.

5. A processor according to any of the preceding

claims wherein the stalling means is arranged such that, when a pipeline generates a stall signal at a stage i, all stages up to and including stage i of that pipeline are stalled.

6. A processor according to any of the preceding claims wherein, when a pipeline generates a stall signal at a stage i, all stages up to and including stage i of that pipeline are stalled on a given clock cycle, and all stages up to and including stage i+m of a pipeline not generating a stall signal are stalled m clock cycles later than said given clock cycle, where m is an integer greater than or equal to 1.

7. A processor according to any of the preceding claims wherein the processor comprises a plurality of pipeline clusters, each cluster comprising a plurality of pipelines.

8. A processor according to claim 7 wherein the stalling means is arranged to stall execution of instructions in pipelines within a cluster in the same clock cycle.

9. A processor according to claim 7 or 8 wherein the stalling means is arranged to stall the execution of instructions in pipelines in a cluster not generating the stall signal at least one clock cycle later than the execution of instructions in pipelines in a cluster generating the stall signal.

10. A processor according to any of the preceding claims wherein, in operation, instructions entering the respective pipelines in parallel exit the pipelines in parallel.

11. A processor comprising a plurality of pipelines, each pipeline having a plurality of pipeline stages for executing an instruction on successive clock cycles, the processor being adapted to allow two instructions which enter and exit respective pipelines in parallel to become out of step with each other for at least one clock cycle.

12. A method of operating a processor, the processor comprising a plurality of pipelines, each pipeline having a plurality of pipeline stages for executing instructions on successive clock cycles, the method comprising generating a stall signal in one of the pipelines, and stalling the execution of instructions in different pipelines in different clock cycles in response to the stall signal.

13. A method according to claim 12 wherein the execution of an instruction in a pipeline not generating the stall signal is stalled at least one clock cycle later than the execution of an instruction in a pipeline generating the stall signal.

14. A method according to claim 13 wherein the stall in the pipeline not generating the stall signal is released at least one clock cycle later than the stall in the pipeline generating the stall signal.

15. A method according to claim 13 or 14 wherein, when a pipeline stage in a first pipeline receives a stall signal from a second pipeline, the execution of instructions in the pipeline stage is not stalled if the pipeline stage stalled in the previous cycle in response to a stall signal generated by the first pipeline.

16. A method according to any of claims 12 to 15 wherein, when a pipeline generates a stall signal at stage i, all stages up to and including stage i of that pipeline are stalled.

17. A method according to any of claims 12 to 16 wherein, when a pipeline generates a stall signal at stage i, all stages up to stage i of that pipeline are stalled on a given clock cycle, and all stages up to stage i+m of a pipeline not generating a stall signal are stalled m clock cycles later than said given clock cycle, where m is an integer greater than or equal to 1.

18. A method according to any of claims 12 to 17 wherein the processor comprises a plurality of pipeline clusters, each cluster comprising a plurality of pipelines, and instructions in pipelines within a cluster are stalled in the same clock cycle.

19. A method according to claim 18 wherein the execution of instructions in pipelines in a cluster not generating the stall signal is stalled at least one clock cycle later than the execution of instructions in pipelines in a cluster generating the stall signal.

20. A method according to any of claims 12 to 19 wherein instructions entering the respective pipelines in parallel exit the pipelines in parallel.

21. A method substantially as described with reference to Figures 1 to 14 of the accompanying drawings.

22. Apparatus substantially as described with reference to and as illustrated in the accompanying

drawings.

ABSTRACT

PROCESSORS AND METHODS OF OPERATING PROCESSORS

Processors comprising a plurality of pipelines are disclosed, each pipeline having a plurality of pipeline stages (140) for executing an instruction on successive clock cycles. The processors allow an instruction in one pipeline to become temporarily out of step with an instruction in another pipeline. This may allow time for a global signal, such as a global stall signal, to be distributed.

(Figure 5)

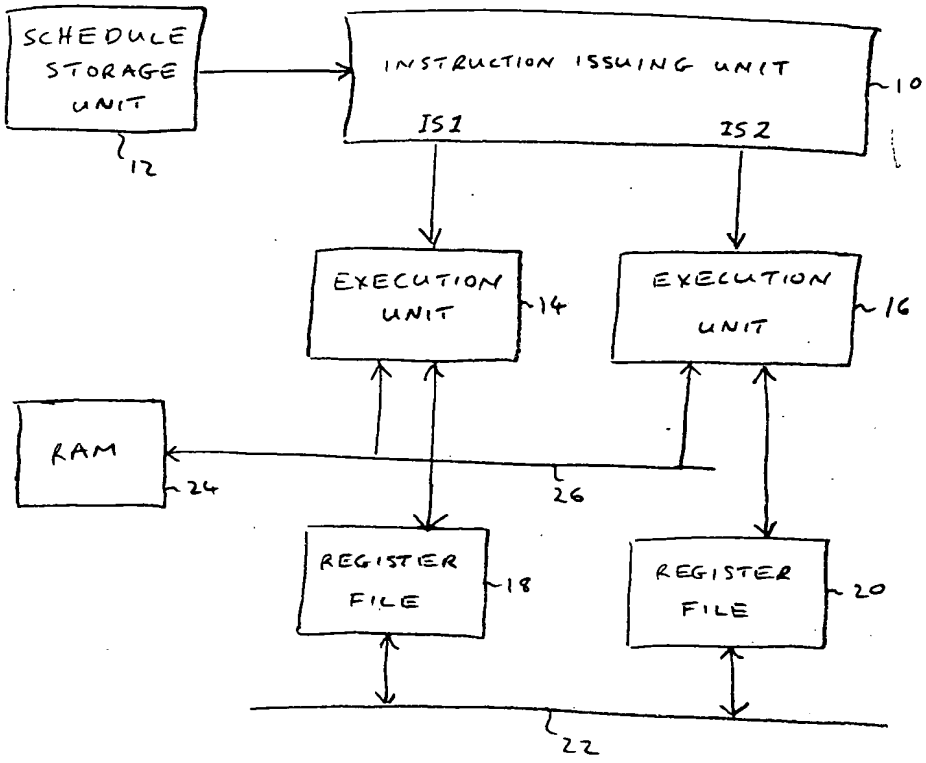


Figure 1

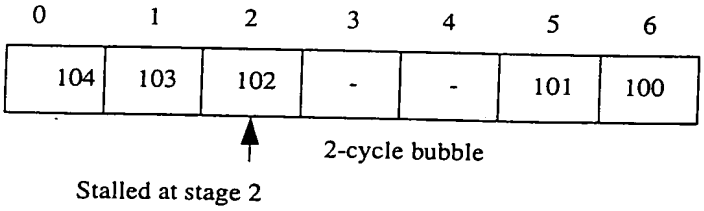
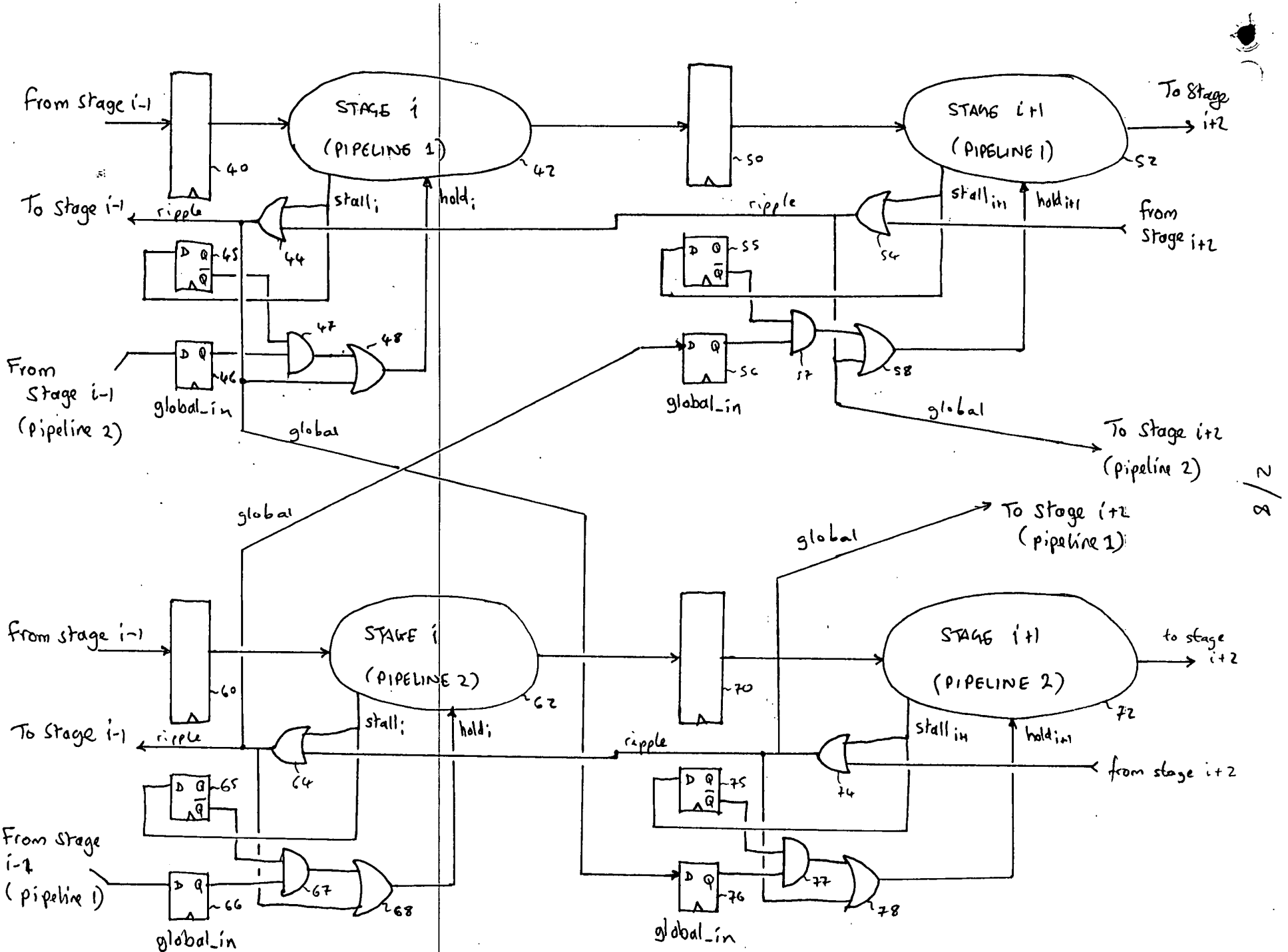


Figure 2



All ⊔ inputs fed by common clock signal.

Figure 3

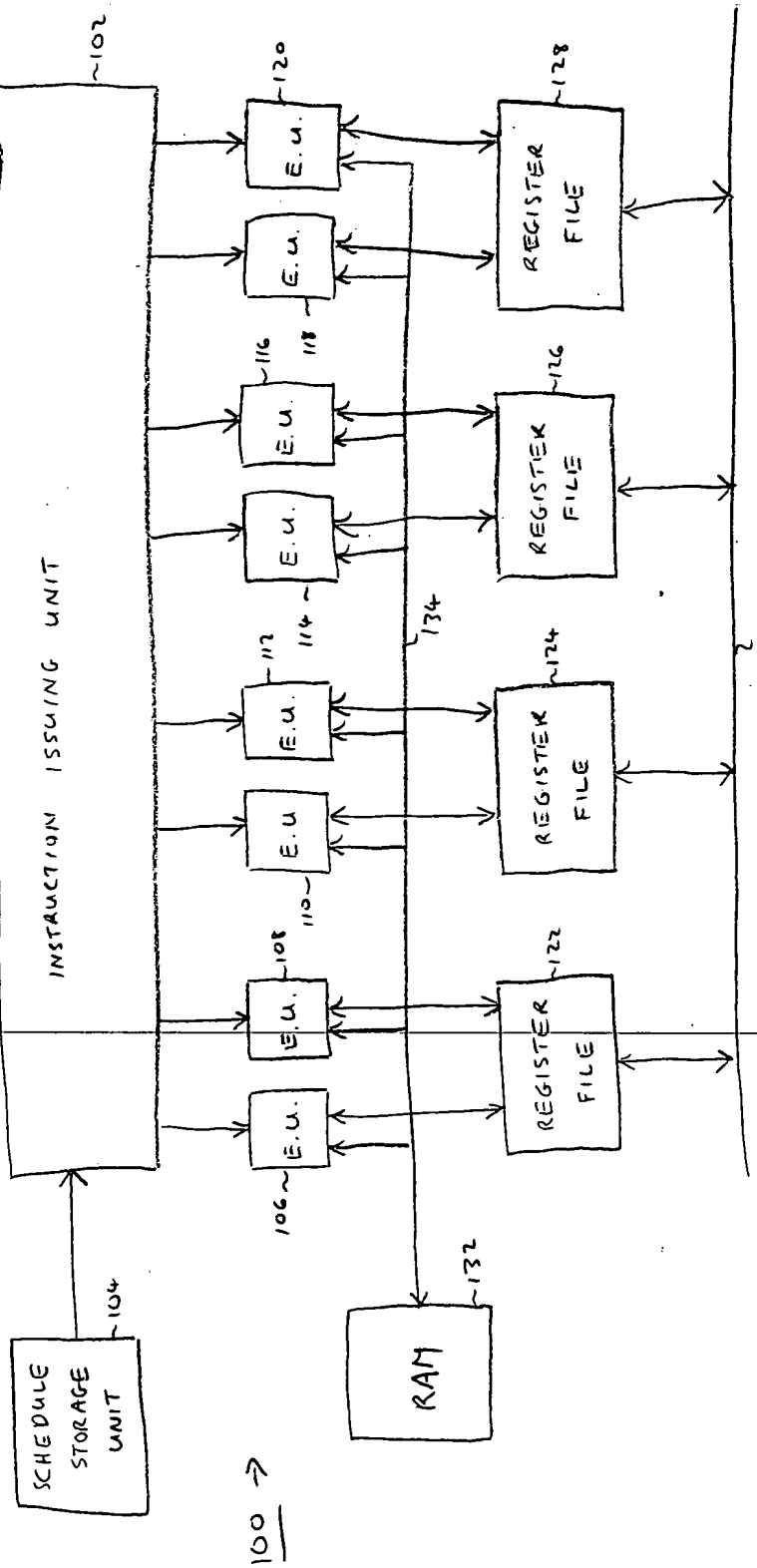

100
SCHEDULE STORAGE UNIT
104
INSTRUCTION ISSUING UNIT
102
RAM
132
134
130
E.U.
106
108
110
112
114
116
118
120
REGISTER FILE
122
124
126
128

Figure 4

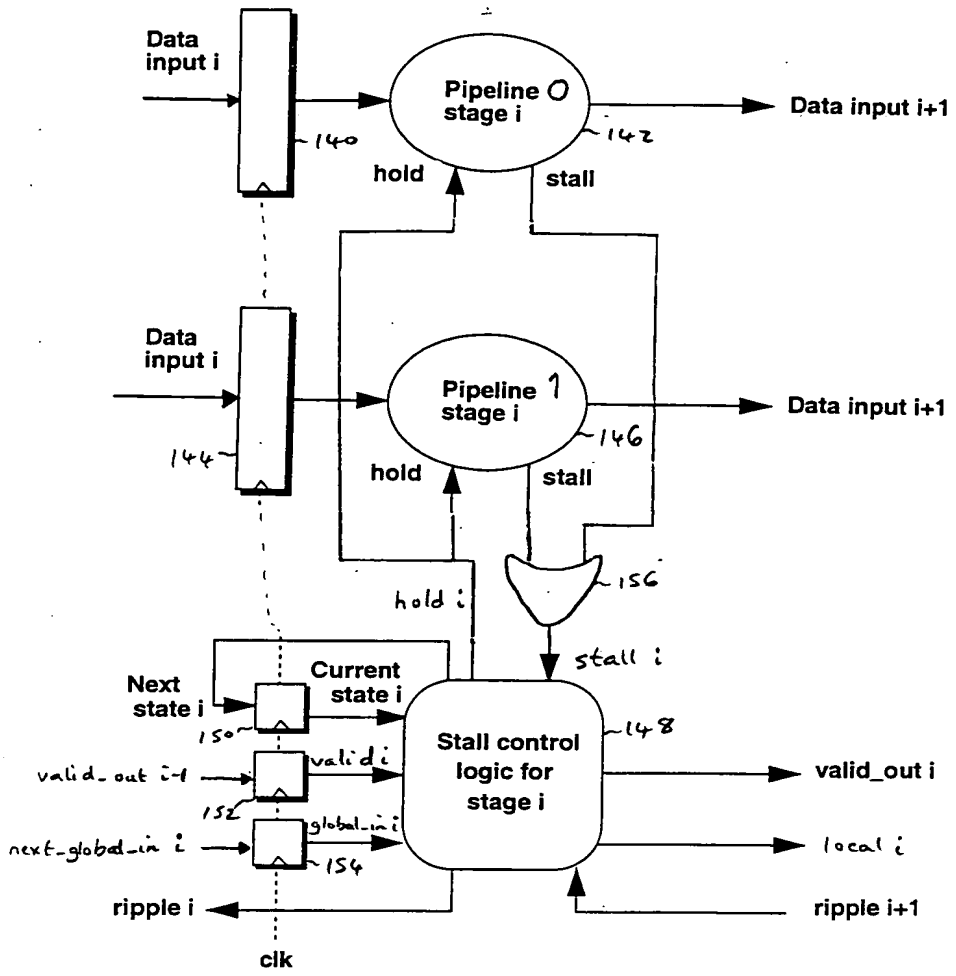


Figure 5

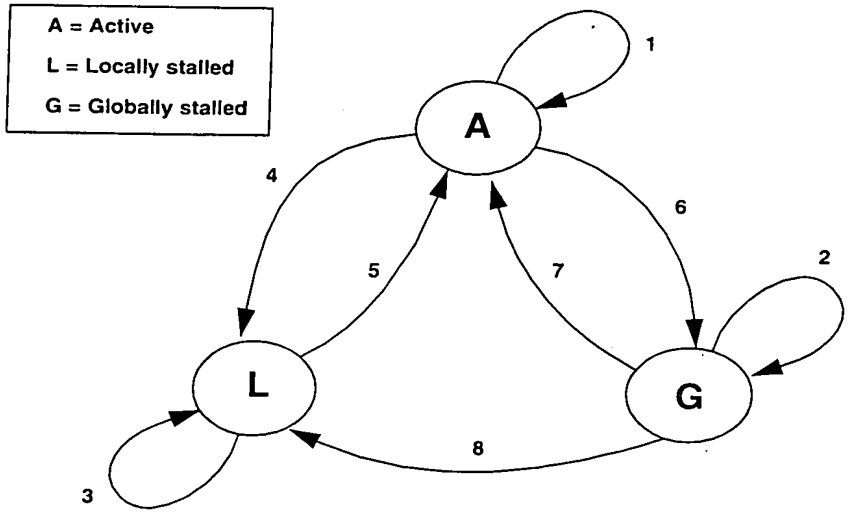


Figure 6

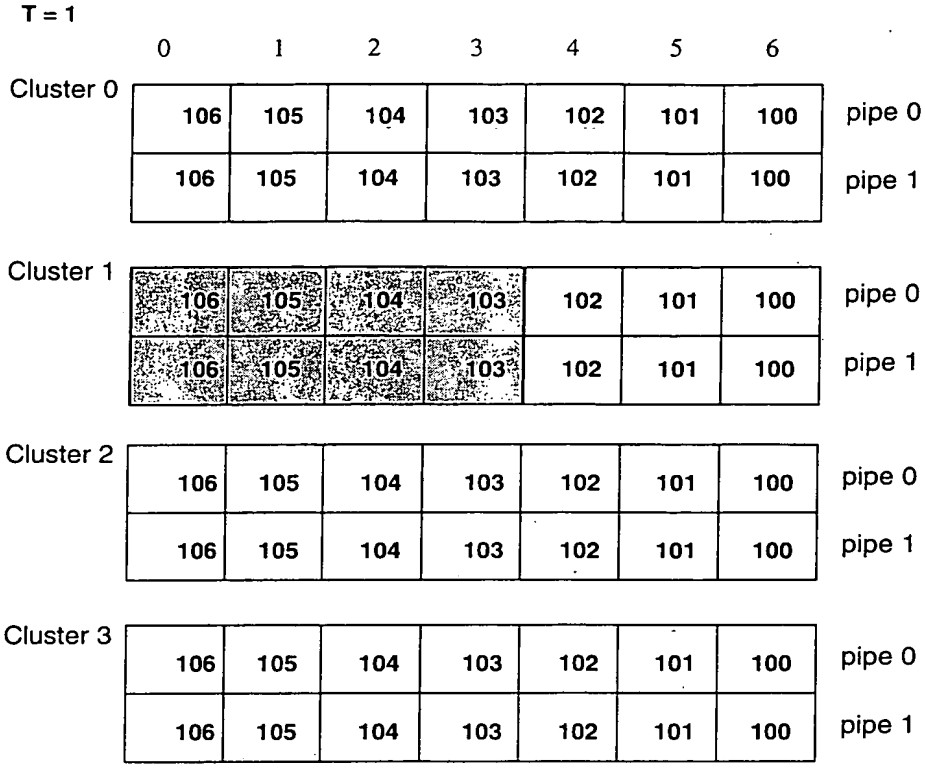


Figure 7

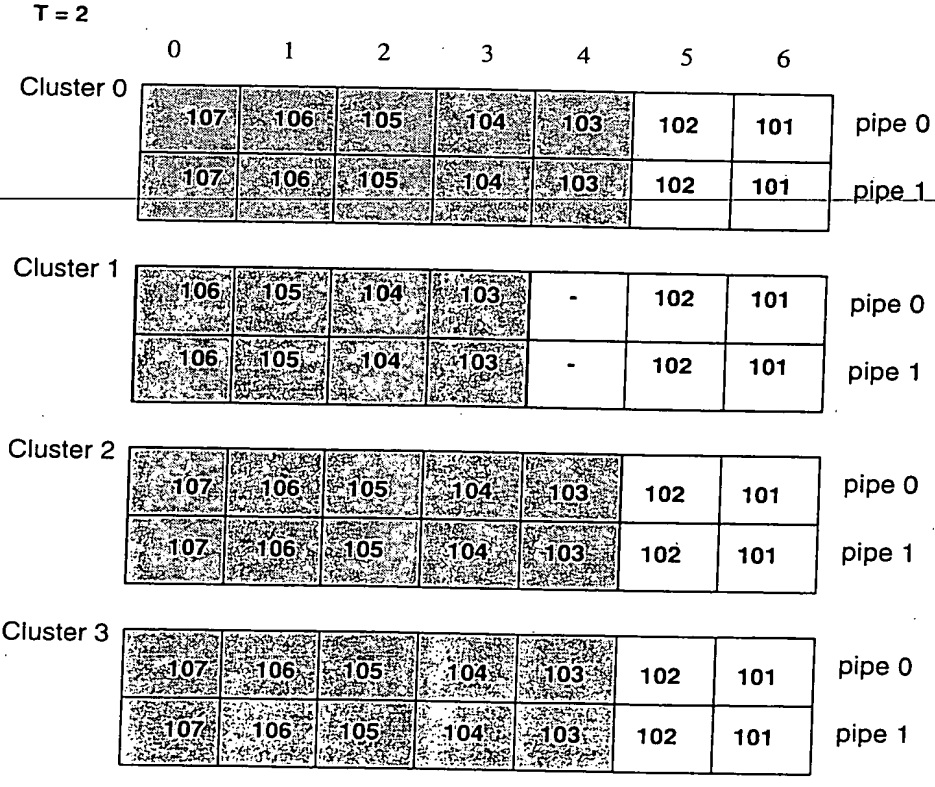


Figure 8

T = 3

	0	1	2	3	4	5	6	
Cluster 0	107	106	105	104	103	-	102	pipe 0
	107	106	105	104	103	-	102	pipe 1
Cluster 1	107	106	105	104	103	-	102	pipe 0
	107	106	105	104	103	-	102	pipe 1
Cluster 2	107	106	105	104	103	-	102	pipe 0
	107	106	105	104	103	-	102	pipe 1
Cluster 3	107	106	105	104	103	-	102	pipe 0
	107	106	105	104	103	-	102	pipe 1

Figure 9

T = 4

	0	1	2	3	4	5	6	
Cluster 0	108	107	106	105	104	103	-	pipe 0
	108	107	106	105	104	103	-	pipe 1
Cluster 1	108	107	106	105	104	103	-	pipe 0
	108	107	106	105	104	103	-	pipe 1
Cluster 2	108	107	106	105	104	103	-	pipe 0
	108	107	106	105	104	103	-	pipe 1
Cluster 3	108	107	106	105	104	103	-	pipe 0
	108	107	106	105	104	103	-	pipe 1

Figure 10

T = 1

	0	1	2	3	4	5	6	
Cluster 0	106	105	104	103	102	101	100	pipe 0
	106	105	104	103	102	101	100	pipe 1
Cluster 1	106	105	104	103	102	101	100	pipe 0
	106	105	104	103	102	101	100	pipe 1
					↑ stall			
Cluster 2	106	105	104	103	102	101	100	pipe 0
	106	105	104	103	102	101	100	pipe 1
			↑ stall					
Cluster 3	106	105	104	103	102	101	100	pipe 0
	106	105	104	103	102	101	100	pipe 1

Figure 11

T = 2

	0	1	2	3	4	5	6	
Cluster 0	107	106	105	104	103	102	101	pipe 0
	107	106	105	104	103	102	101	pipe 1
Cluster 1	106	105	104	103	—	102	101	pipe 0
	106	105	104	103	—	102	101	pipe 1
Cluster 2	106	105	—	104	103	102	101	pipe 0
	106	105	—	104	103	102	101	pipe 1
Cluster 3	107	106	105	104	103	102	101	pipe 0
	107	106	105	104	103	102	101	pipe 1

Figure 12

$T = 3$

	0	1	2	3	4	5	6	
Cluster 0	107	106	105	104	103	–	102	pipe 0
	107	106	105	104	103	–	102	pipe 1
Cluster 1	107	106	105	104	103	–	102	pipe 0
	107	106	105	104	103	–	102	pipe 1
Cluster 2	107	106	105	104	103	–	102	pipe 0
	107	106	105	104	103	–	102	pipe 1
Cluster 3	107	106	105	104	103	–	102	pipe 0
	107	106	105	104	103	–	102	pipe 1

Figure 13

$T = 4$

	0	1	2	3	4	5	6	
Cluster 0	108	107	106	105	104	103	–	pipe 0
	108	107	106	105	104	103	–	pipe 1
Cluster 1	108	107	106	105	104	103	–	pipe 0
	108	107	106	105	104	107	–	pipe 1
Cluster 2	108	107	106	105	104	103	–	pipe 0
	108	107	106	105	104	103	–	pipe 1
Cluster 3	108	107	106	105	104	103	–	pipe 0
	108	107	106	105	104	103	–	pipe 1

Figure 14